WHAT IS CLAIMED IS:

1. A code preparation method comprising:
identifying at least one operation in first executable instance of code;
executing the first executable instance and responsive to detection of an execution event, associating a corresponding execution characteristic with a corresponding identified one of the operations; and
preparing a second executable instance of the code based, at least in part, on the association between the execution characteristic and the identified operation.

2. The method of claim 1,
wherein the operation identification is consistent between the first executable instance and the preparation of the second executable instance.

3. The method of claim 2,
wherein the consistency of operation identification is maintained from preparation of the first executable instance to preparation of the second executable instance.

4. The method of claim 1,
wherein same unique identification numbers are assigned to corresponding operations of the first executable and the second executable.

5. The method of claim 4,
wherein the execution characteristic is associated with the unique identification number.

6. The method of claim 4,
wherein the unique identification numbers and their assignment to operations are maintained throughout any optimizations or code transformations performed in preparation of the first executable.

7. The method of claim 6,
wherein the maintenance of the unique identification number assignments include further assigning the unique identification number to a copy when an operation is copied as part of a code transformation or optimization.

8. The method of claim 6,
wherein the maintenance of the unique identification number assignments includes removing an assignment when the assigned operation is removed as part of a code transformation or optimization.

9. The method of claim 1,
wherein the associating of the corresponding execution characteristic includes encoding aggregated hardware event information in an extended definition of an instruction instance for use in the preparation of the second executable instance.

10. The method of claim 1,
wherein the identified operation is a memory access instruction.

11. The method of claim 1,
wherein the execution characteristic includes a cache miss likelihood.

12. The method of claim 1,
wherein the preparation includes inserting one or more prefetch operations in the code prior to the identified operation to exploit latency provided by servicing of a cache miss by the identified operation.

13. The method of claim 1, further comprising:
preparing the first executable instance.

14. The method of claim 13,
wherein the preparation of the first executable instance includes substantially all optimizations operative in the preparation of the second executable.

15. The method of claim 14,
wherein execution of the first executable instance corresponds substantially with execution of an executable instance of code prepared without the identifying.

16. The method of claim 14,
whereby execution of the first executable instance sufficiently corresponds to that in an expected execution environment, so that the execution characteristic is applicable to the preparation of the second executable.

17. The method of claim 13,
wherein the preparation of the first executable instance forgoes certain optimizations performed, after use of the association between the execution characteristic and the identified instruction, by the further preparing.

18. The method of claim 13,
wherein the preparation of the first executable instance includes compilation of the code.

19. The method of claim 1,
wherein both the first and the second executable instances are compiled instances of the code.

20. The method of claim 1,
wherein the second executable instance is an optimization of the first executable instance.

21. The method of claim 1,
wherein the preparing includes optimizations forgone in the first executable instance.

22. The method of claim 1,
wherein the preparation of the second executable instance includes optimizations forgone in preparation of the first executable instance.

23. The method of claim 1,
wherein at least the preparing is performed by an optimizing compiler.

24. The method of claim 1,
wherein at least the preparing is performed by a binary translator.

25. The method of claim 1,
wherein at least the preparing is performed by a binary rewriter.

26. The method of claim 1,
wherein at least the preparing is performed by a binary optimizer.

27. The method of claim 1,
wherein at least the preparing is performed by a just-in-time (JIT) compiler.

28. The method of claim 1,
wherein the associating of the corresponding execution characteristic includes aggregating contributions of plural instances of the execution event.

29. The method of claim 1,
wherein the associating of the corresponding execution characteristic includes backtracking from a point in the code that coincides with delayed detection of the execution event.

30. The method of claim 1,
wherein the associating of the corresponding identified one of the operations includes reading or receiving a computer readable encoding of an event profile.

31. The method of claim 1,
wherein the associating of the corresponding execution characteristic includes reading or receiving a computer readable encoding of an event profile.

32. The method of claim 1, further comprising:
preparing the second executable instance as a computer program product for distribution, transmission or execution.

33. The method of claim 33,
wherein the computer program product is encoded in one or more computer readable media selected from the set of a disk, tape or other magnetic, optical, semiconductor or electronic storage medium and a network, wireline, wireless or other communications medium.

34. An optimizing compiler that prepares a second executable instance of computer program code including optimizations in addition to those of a previously prepared first executable instance thereof, wherein the additional optimizations include performing one or more transformations based on run-time information from execution of the first executable instance, wherein consistency of instruction identification is maintained from preparation of the first executable instance to preparation of the second executable instance.

35. The method of claim 34,
wherein same unique identification numbers are assigned to corresponding operations of the first executable and the second executable.

36. The method of claim 35,
wherein the unique identification numbers and their assignment to operations are maintained throughout any optimizations or code transformations performed in preparation of the first executable.

37. The method of claim 36,
wherein the maintenance of the unique identification number assignments include further assigning the unique identification number to a copy

when an operation is copied as part of a code transformation or optimization.

38. The method of claim 36,
wherein the maintenance of the unique identification number assignments includes removing an assignment when the assigned operation is removed as part of a code transformation or optimization.

39. The optimizing compiler of claim 34,
wherein the transformations include insertion of one or more prefetch instructions.

40. The optimizing compiler of claim 34,
wherein the transformations include insertion of one or more non-faulting loads.

41. The optimizing compiler of claim 34,
wherein selection of optimizations performing in the preparation of the first executable instance is biased toward collection of data.

42. The optimizing compiler of claim 34,
wherein the additional optimizations performing in the preparation of the second executable instance are biased toward obtaining improved performance based on the run-time information.

43. The optimizing compiler of claim 34,
wherein transformations include insertion of instructions into the second executable instance to reduce latency of memory access operations that, based on the run-time information, are likely to miss in a cache.

44. The optimizing compiler of claim 34,
wherein the optimizing compiler prepares the second executable instance, but not the first.

45. The optimizing compiler of claim 34,

wherein the optimizing compiler also prepares the first executable instance of computer program code.

46. The optimizing compiler of claim 34, embodied as part of a binary translator.

47. The optimizing compiler of claim 34, embodied as part of a binary rewriter.

48. The optimizing compiler of claim 34, embodied as part of a binary optimizer.

49. The optimizing compiler of claim 34, embodied as a just-in-time (JIT) compiler.

50. The optimizing compiler of claim 34,

wherein first and second executions of the optimizing compiler respectively provide the first and second executable instances; and

wherein the transformations are performed in addition to optimizations coextensive with those performed in the first executable instance.

51. The optimizing compiler of claim 34,

wherein the optimizing compiler identifies one or more memory access instructions in the first executable instance of the computer program code; and

wherein the run-time information encodes respective execution characteristics for respective ones of the identified memory access instructions.

52. The optimizing compiler of claim 34,

wherein collection of the run-time information includes aggregation of execution event information and association of the aggregated information with memory access instructions identified in the first executable instance of the computer program code.

53. The optimizing compiler of claim 34,
encoded in one or more computer readable media selected from the set of a disk, tape or other magnetic, optical, semiconductor or electronic storage medium and a network, wireline, wireless or other communications medium.

54. A method of optimizing code for an execution environment in which a possibility of processor or pipeline stall latency exists for particular instructions thereof, the method comprising:
identifying the particular instructions in a first executable instance of the code;
associating a characterization of stall likelihood with respective ones of the particular instructions based on at least one execution of the first executable instance; and
inserting behind respective ones of the particular instructions, one or more pre-executable portions of the particular instructions selected to reduce stall latency thereof based on the respective associated characterization of stall likelihood.

55. The method of claim 54,
wherein the pre-executable portions include prefetch instructions.

56. The method of claim 54,
wherein the pre-executable portions include non-faulting loads.

57. The method of claim 54,
wherein the particular instructions are memory access instructions and the associated characterizations are of cache miss likelihood.

58. The method of claim 54, further comprising:
executing the first executable instance of the code to obtain the characterization of stall likelihood for the particular instructions.

59. The method of claim 54, further comprising:
preparing a computer program product encoding a second executable instance of the code that includes the inserted prefetch instructions.

60. The method of claim 54, further comprising:
preparing the first executable instance of the code.

61. A computer program product encoded in one or more computer readable media, the computer program product comprising:
a first execution sequence; and
an information encoding associating an execution event with at least some operation of the first execution sequence, the associated execution event based at least in part on an execution profile of the first execution sequence of operations, wherein consistency of the association is maintained from preparation of the first executable instance for preparation of a second executable instance.

62. The computer program product of claim 61,
wherein the execution event is a cache miss likelihood.

63. The computer program product of claim 61,
wherein the associated operation is a memory access operation.

64. The computer program product of claim 61,
employed in an data structure of an optimizing compiler in preparation of an optimized instance of the execution sequence of operations, wherein the optimized instance includes one or more prefetch operations placed before particular ones of the memory access operations for which the associated information encoding indicates a cache miss likelihood.

65. The computer program product of claim 61,
wherein the one or more computer readable media are selected from the set of a disk, tape or other magnetic, optical, semiconductor or electronic

storage medium and a network, wireline, wireless or other communications medium.

66. An apparatus comprising:
means for identifying in a first executable instance of computer program code certain operations thereof for inclusion in an execution profile;
means for collecting the execution profile; and
means for preparing a second executable instance of the computer program code, wherein consistency of the identifying is maintained for operations thereof corresponding to the certain operations such that the corresponding certain operations are relatable to the execution profile.

67. The apparatus of claim 66,
wherein the identifying includes producing a table of tags and operation addresses.

68. The apparatus of claim 66,
wherein information for the identifying is encoded in a file or communications channel read by the means for collecting.

69. The apparatus of claim 66, further comprising:
means for preparing the first executable instance of the computer program code.